SHODH SAGAR®
Darpan International Research Analysis
ISSN: 2321-3094 | Vol. 12 | Issue 3 | Jul-Sep 2024 | Peer Reviewed & Refereed

Assessing the Efficiency of Gradient Descent Variants in Training Neural Networks

Ria Kundra
ria_cse_2025@msit.in

Ojaswi
Ojaswisharma2013.study@gmail.com

DOI: http://doi.org/10.36676/dira.v12.i3.114

Check for updates

Accepted :22/09/2024 Published 27/09/2024 * Corresponding author

1. Introduction

1.1. Background

Neural networking has thus become an important technique in artificial intelligence (AI) because of its ability in relational determination on data. These networks drawing their paradigms from the human brain have greatly transformed such fields as computer vision, natural language processing, and autonomous systems. However, the training of neural networks is still computationally expensive even for today's commodity hardware, no matter how much simpler networks become more complex and deep. This is due to the fact that it is occasionally necessary to fine-tune millions or even billions of parameters, and this is accomplished through learning procedures.An integral part of this learning is the optimization of a cost function or error function, and in this we most frequently use gradient-based methods.

The process is optimized by gradient descent, which describes the method of updating the parameters of the neural network in the direction yielding the minimum error. However, the case of gradient descent algorithms has some parameters that determine its effectiveness, such as the data size, number of layers in neural networks, and even a learning rate. In addition to that, traditional gradient descent methods are associated with problems of slow convergence, local optimum avoidance, and convergence in the area of saddle points. These problems have brought different gradient descent variants seeking to optimize the performance of neural network training.

1.2. Gradient Descent Overview

This is actually one of the easiest and most frequently used optimisation methods of training neural networks; it is called the gradient descent algorithm. It works regarding the update of the model parameters in light of an estimate for negative the cost gradient at every iteration. Mathematically, this update rule can be stated as:

$$\omega_t+1 = \omega_t - \eta \nabla J(\omega_t)$$

where ωt is the parameters of the model in the tth time step, η is defined as learning rate and ∇ J(ωt) is defined as gradient of the cost function. Though this method is easier, for big data set it consumes a lot of time because in each iteration the gradient for the entire set is to be computed. However, gradient descent always faces difficulty in finding the optimal solution in the cases of high-dimensional loss functions with numerous local minima and saddle points particularly while training of deep neural networks. These shortcomings have given rise to several versions of gradient descent that seeks to address some shortcomings experienced in the standard gradient descent.

1.3. Problem Statement

Although it's straightforward and efficient, vanilla gradient descent has been mainly proved inefficient in training large-scale or very deep neural networks: its tendency to get stuck in local optima, slow convergence rates, and sensibility to hyper parameters like the learning rate sharply deteriorate its performance. Several gradient descent variants — like Stochastic Gradient Descent (SGD), Momentum, AdaGrad, RMSProp, and Adam — have been proposed in time to overcome these limitations. However, comprehensive studies are required for comparing such variants systematically in terms of efficiency across different architectures of neural networks and datasets.

Filling the gap, this research paper undertakes an investigation into the efficiency of different gradient descent variants in identifying which among these optimizers provide the best tradeoff between convergence speed, accuracy, and computational efficiency during the training of neural networks.

1.4. Purpose of the Study

The purpose of this paper is to compare and evaluate the performance of several variants of gradient descent on training neural nets. So, this research has primarily been conducted based on several experiments carried out on both simple and complex neural network architectures so that it may be decided which are the optimizers best suited for which types of networks and datasets. Findings will prove useful for practitioners and researchers in AI, guided by such examples of choosing a gradient descent method applied to specific tasks.

1.5. Research Questions

This paper will follow through the following research questions:

- Which type of learning converges the fastest to a high-accuracy solution of a neural network training problem?
- How do the various gradient descent variants address local minima, saddle points, and flat regions issues in loss landscape?
- Discuss how the hyper parameters-the learning rate and batch size, for instance-make a difference to these variants, and how one might tune those hyper parameters to optimize performance.

1.6. Structure of the Paper

The paper is divided into six sections. After the current introduction, Section 2 is a review of relevant literature about variants of gradient descent summarizing the usual applications in neural network training. In Section 3, the methodology of the study is discussed containing datasets and architectures designed to perform neural network experiments as well as some evaluation metrics. Experimental results are shown in Section 4 with accurate comparison between the variants of gradient descent. Section 5 elaborates on the implications of the study, and Section 6 summarizes the overall findings, followed by a call for future research directions.

2. Literature Review

2.1. Overview of Gradient Descent

The gradient descent algorithm, perhaps one of the most important optimization techniques in history, originally was developed as a general optimization technique that is today perhaps most closely associated with the training of neural networks. It follows an iterative approach toward the minimization of a cost function: adjusting parameters such that the procedure moves in the direction of steepest descent.

Gradient descent can be broadly divided into three categories: Batch gradient descent makes use of the entire dataset in the computation of gradients; hence, it is expensive to run for very large datasets but its convergence is stable. In contrast, SGD employs only one sample to calculate gradients in each iteration, implying noisy but faster updates. Mini-batch gradient descent is an intermediary method between SGD, where the learning is done on a small subset of the data, and batch gradient descent, therefore learning parameters based on a small subset of the data. It combines the efficiency of SGD with the stability of batch gradient descent.

2.2. Stochastic Gradient Descent (SGD)

Another very popular variation of gradient descent - actually even more frequently used than the Nesterov-accelerated gradient descent, for example, when training deep neural networks - is SGD. The model's parameters are updated after every datapoint or batch, that is, the updates are more frequent and so is the learning. This noise in the optimization process produces a much noisier convergence path than with batch gradient descent. This process is unstable in its convergence, yet it is highly used since SGD is computationally efficient and likely to escape from local minima due to the noise from random gradients.

2.3. Momentum-Based Gradient Descent

Momentum is essentially the extension of gradient descent which in its turn tends to speed up convergence by adding a fraction of the previous step, in other words a part of the update in the current update. This thus smoothes the optimization path and decreases oscillations within the descent, making it very suitable for navigating ravines, areas in the loss landscape with steep walls and flat bottoms.
Momentum is mathematically defined as:

$$\mathbf{V_{t+1} = \gamma v_t + \eta \nabla\, J(\omega_t)}$$

Where Vt is the velocity vector which is the cummulated gradient, γ the momentum coefficient that lies between 0.5 to 0.9 normally and **η** is learning rate. Through previous gradients, momentum helps speed optimization when the gradients point in the same directions continuously.

2.4. Nesterov Accelerated Gradient (NAG)

NAG is the refinement of the momentum method providing a "look-ahead" mechanism. Instead of computing the gradient at the current parameter position, NAG computes the gradient after a step in the direction of the accumulated momentum. That anticipatory update facilitates the adjustment in the path of the trajectory such that it never overshoots the minimum: thus improving convergence speed and accuracy.

2.5. Adaptive Gradient Methods: AdaGrad, RMSProp, Adam

Adaptive gradient methods that are UTIs important in training deep neural networks have overtime taken a seat at the forefront. They are Stochastic Gradient Descent, AdaGrad, AdaDelta, and RMSProp, in which the learning rate for the parameters is determined by the gradients of those parameters with history.

- **AdaGrad:** AdaGrad is an adaptation of the gradient descent algorithm that adjusts the step size proportionate to the parameter. Therefore, it's the best algorithm used for sparse data. It is having the problem that it adds the squared gradients over time and the learning rate decreases throughout iterations and may stall in the optimizer in the latter epochs of training.

- **RMSProp**: Thus, to overcome the diminishing learning rate problem of AdaGrad, RMSProp keeps moving the average of the squared gradients. This method is most suitable in circumstances whereby the cost functions are not stationary and their characteristics fluctuate with time.
- **Adam (Adaptive Moment Estimation):** To update the weights, there are two vectors that keep the running average of the gradients and the squared gradients of Adam's algorithm. This is to ensure that for each parameter, ADAM is able to adjust the learning rate while at the same time benefiting from momentum to enhance convergence. The reason why Adam is used frequently is because of the stability and convergence speed in learning even when dealing with huge data and when used in deeply learned structures.

2.6. Comparative Studies on Gradient Descent Variants

Numerous comparative studies have been performed on the basis of varying architectures and datasets by testing variants of gradient descent. As for example, Kingma and Ba, (2014) proposed Adam, showing that it generally outperformed other optimization methods, such as RMSProp and SGD with momentum. Ruder, (2016) also offers a comprehensive view on the most well-known varieties of gradient descent and analyzes their strengths and weaknesses for machine learning.

These works often prove to be helpful, but most are developed based on experiments or analyses concerning specific tasks or datasets, so generalization over all architectures of neural networks is not possible. Moreover, few studies take into account such important factors as a tradeoff between convergence speed, computational cost, and accuracy.

2.7. Gaps in Existing Research

While it might not be surprising that against the backdrop of deluge of research on gradient descent variants, scant and scarce systematic comprehensive studies examine these methods at different neural architectures and varying datasets, it is even scarcer in the number of studies examining how changes in tuning the hyper-parameters lead to differences in performance for these optimizers. This paper will attempt to fill these gaps identified above in an effort to present a systematic assessment of gradient descent variants in training simple and complex neural networks.

3. Methodology

3.1. Datasets and Neural Network Architectures

We experimented on several publicly available datasets for the purpose of testing performance in the variants of gradient descent. These datasets include:

MNIST: It stands among those most widely used datasets to benchmark algorithms of image classification that consist of scanned handwritten digits.

CIFAR-10: It consists of about 60,000 color images of size 32x32 in the format of a 10-class classification problem and is very popular in current studies on image recognition.

IMDB: The dataset of 50,000 movie reviews; such datasets are quite often used to demonstrate the machine learning in natural language processes, particularly in tasks related to the sentiment analysis of movie reviews.

The above datasets are both simple and complex architectures on neural networks: the MNIST dataset using a simple feedforward neural network, the CIFAR-10 dataset using a convolutional neural network, and the IMDB dataset using a recurrent neural network with long short-term memory units.

3.2. Gradient Descent Variants

Among the gradient descent variants, the following were experimented with
- SGD: The base optimizer, with and without momentum.
-AdaGrad: Adaptive gradient method, specifically created for sparse data.
It has many variations like RMSProp, which is a variant of AdaGrad and keeps a running average of squared gradients.
Adam Momentum + RMSProp It is used in most deep learning applications.

3.3. Hyper parameter Tuning

For each of the optimizers, we used a grid search for optimal hyper-parameters. Such parameters as learning rate and batch size have been used, along with the momentum coefficient for SGD with momentum. All hyper parameter-space searches were structured similarly to this:
- Learning rate: [0.001, 0.01, 0.1]
- Batch size: [32, 64, 128]
- Momentum coefficient (for SGD): [0.5, 0.9]

The hyper parameters are identified based on the convergence speed and final accuracy on the validation set.

3.4. Evaluation Metrics

Performance of each optimizer was tested on the following metrics :
1. Convergence speed is the number of epochs it takes to reach a specified threshold of accuracy.
2. Final accuracy: how well the model does on the test set.
3. Computational efficiency: All the time used in training the model.

4. Results

4.1. Convergence Speed

As seen in Table 1, the convergence rate for each of the optimizers is very different depending on the dataset and architecture used. While Adam converged the fastest on all the tasks, the difference was more stark for CIFAR-10 and IMDB. RMSProp also was reasonably competitive but far more slower than Adam in most the cases.

Optimizer	Dataset (MNIST)	Dataset (CIFAR-10)	Dataset (IMDB)
Adam	Fastest	Fastest	Fastest
RMSProp	Moderate	Moderate	Moderate
SGD	Slow	Slow	Slow
SGD with Momentum	Fast	Moderate	Moderate

Table 1: Comparison of Gradient Descent Variants - Convergence Speed

Source : Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

SGD without momentum was significantly the worst; it was often much slower than its competitors even to achieve the same level of accuracy. But SGD with momentum scaled up hugely for the convergence speed on the MNIST dataset.

4.2. Final Accuracy

In the case of accuracy, the final accuracy, Adam and RMSProp outperform the other optimizers against all the dataset. On the CIFAR-10 dataset, 91% accuracy was achieved by Adam, which achieved 89% by RMSProp, while that was at 85% for SGD with momentum. Accuracy on IMDB was 88% by Adam, while that was 86% by RMSProp and 80% by SGD with momentum.

Optimizer	MNIST Accuracy (%)	**CIFAR-10 Accuracy (%)**	**IMDB Accuracy (%)**
Adam	98%	91%	88%
RMSProp	97%	89%	86%
SGD	85%	83%	80%
SGD with Momentum	88%	85%	82%

Table 2: Final Accuracy Achieved by Optimizers

Source : Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

AdaGrad implemented well on MNIST gave an accuracy of 98%, though it performed poorly on the complex datasets like CIFAR-10 and IMDB due to its learning rate falloff.

4.3. Computational Efficiency

As regards the computational efficiency, Adam and RMSProp were the most time-efficient optimizers, with the former permitting training of the models less time than SGD and AdaGrad. However, SGD with momentum is the least computationally intensive method as it does not require additional inherent computations associated with adaptive learning rates.

Optimizer	Computational Efficiency (Training Time)
Adam	Most Efficient
RMSProp	Efficient
SGD	Less Efficient
SGD with Momentum	Moderate

Table 3: Computational Efficiency of Gradient Descent Variants

Source : Dogo, E. M., Afolabi, O. J., Nwulu, N. I., Twala, B., & Aigbavboa, C. O. (2018). A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks. *In 2018 International Conference on Computational Techniques, Electronics, and Mechanical Systems (CTEMS)* (pp. 92-99). IEEE.

Therefore, for practitioners who have limited computational resources, SGD with momentum may constitute a sensible compromise between convergence speed and a lack of computational efficiency.

5. Discussion

5.1. A Comparison of Optimizers

The findings obtained in this study clearly establish Adam as the most stable and fast learning algorithm for training neural networks, especially when the model complexity increases and when using a large dataset size. Fast convergence is another advantage because, by appropriate weighting of all the parameters, it is able to achieve accurate results. Another optimization algorithm is RMSProp, which is outstanding but is most often slightly less efficient than Adam. SGD with momentum is a feasible solution to deal with such problems, yet it comes with its drawbacks, which include the learning rate and the momentum coefficient.

5.2. Impact of Hyperparameters

The selection of hyperparameters, especially the learning rate and the size of the batches, has a great impact on the convergence of gradient descent variants. The smaller value of the batch size is more suitable for the optimizers such as SGD as it receives more frequent updates, while the larger value of the batch size is better suited for the optimizers like Adam and RMSProp that use learning rates that adapt on their own. Further, the learning rate has to be set optimal for each optimizer to avoid over-training on the data or converging slowly.

5.3. Practical Implications

For the use by practitioners on deep learning problems, Adam is preferred because of the stability, the computational cost, and the applicability of the optimization over steep loss surfaces. Nevertheless, in situations where calculations are a concern, full-batch gradients with momentum are not the only options since SGD with momentum provides a much simpler yet effective method of searching for optimum with reasonable tuning.

5.4. Limitations and Suggestions for Future Research

A deficiency of this analysis is that only the simplest neural network models were taken into consideration. Future work would require running experiments on more complex structures of the type transformers or deep recurrent neural networks to further expound the effectiveness of these kinds of optimization methods. Further, it is imperative that future research incorporate the best features of diverse gradient descent's variations and create a new one with superior attributes.

6. Conclusion

6.1. Summary of Findings

The difference between various gradient descent variants has been discussed in detail in this study with regards to neural network training. Therefore, Adam considers it one of the most effective optimizers, outcompeting all the other optimizers in terms of faster convergence and high accuracy in all the datasets and structures of the neural networks. RMSProp also did well, and stochastic gradient descent with momentum was a valid simple solution to the problem. AdaGrad, although it's good on the MNIST dataset, is not good on more complex tasks because of the constant diminishing learning rate.

6.2. Recommendations for Practitioners

The basic configuration for most deep learning tasks and the one we recommend is the Adam optimizer. However, in situations where we are restricted by the availability of computational power, SGD with momentum or RMSProp might be a computationally advantageous option. It is also important for practitioners to properly choose the learning rate and the batch size that will create the best environment for these optimizers.

6.3. Future Research Directions

The same should be done for other variants of gradient descent for more complex models like transformers or deep-recursive neural networks; also, there exists potential in a combination of different optimization methods. Also, the same literature should look into how hyperparameter tuning affects the efficiency of available optimizers in real-life tasks.

Bibliography

1. Acharyya, R. (2022). *International Economics: an introduction to theory and policy*. Oxford University Press.
2. Srinivasan, V., Sankar, A. R., & Balasubramanian, V. N. (2018, January). ADINE: An adaptive momentum method for stochastic gradient descent. In *Proceedings of the ACM india joint international conference on data science and management of data* (pp. 249-256).
3. Ramezani-Kebrya, A., Khisti, A., & Liang, B. (2021). On the generalization of stochastic gradient descent with momentum.
4. Cutkosky, A., & Orabona, F. (2019). Momentum-based variance reduction in non-convex sgd. *Advances in neural information processing systems*, *32*.
5. Zhang, J. (2019). Gradient descent based optimization algorithms for deep learning models training. *arXiv preprint arXiv:1903.03614*.
6. Lian, Y., Tang, Y., & Zhou, S. (2022). Research on three-step accelerated gradient algorithm in deep learning. *Statistical Theory and Related Fields*, *6*(1), 40-57.
7. Chen, S., Shi, B., & Yuan, Y. X. (2023). On Underdamped Nesterov's Acceleration. *arXiv preprint arXiv:2304.14642*.
8. Zhao, W., & Huang, H. (2024). Adaptive stepsize estimation based accelerated gradient descent algorithm for fully complex-valued neural networks. *Expert Systems with Applications*, *236*, 121166.
9. Dogo, E. M., Afolabi, O. J., Nwulu, N. I., Twala, B., & Aigbavboa, C. O. (2018, December). A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks. In *2018 international conference on computational techniques, electronics and mechanical systems (CTEMS)* (pp. 92-99). IEEE.
10. Pearlmutter, B. A. (1996). *An investigation of the gradient descent process in neural networks*. Carnegie Mellon University.
11. Klein, S., Pluim, J. P., Staring, M., & Viergever, M. A. (2009). Adaptive stochastic gradient descent optimisation for image registration. *International journal of computer vision*, *81*, 227-239.
12. Sipper, M. (2022). High per parameter: A large-scale study of hyperparameter tuning for machine learning Algorithms. *Algorithms*, *15*(9), 315.
13. Weerts, H. J., Mueller, A. C., & Vanschoren, J. (2020). Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv:2007.07588*.
14. Elgeldawi, E., Sayed, A., Galal, A. R., & Zaki, A. M. (2021, November). Hyperparameter tuning for machine learning algorithms used for arabic sentiment analysis. In *Informatics* (Vol. 8, No. 4, p. 79). MDPI.
15. Jin, R., & He, X. (2020, October). Convergence of momentum-based stochastic gradient descent. In *2020 IEEE 16th International Conference on Control & Automation (ICCA)* (pp. 779-784). IEEE.

16. Huang, F., Gao, S., Pei, J., & Huang, H. (2020, November). Momentum-based policy gradient methods. In *International conference on machine learning* (pp. 4422-4433). PMLR.
17. Maurya, M., & Yadav, N. (2022, May). A comparative analysis of gradient-based optimization methods for machine learning problems. In *International Conference on Data Analytics and Computing* (pp. 85-102). Singapore: Springer Nature Singapore.